Instance Monitor™

Tuning Guide

Version 1.0

QUEST SOFTWARE

Quest Software
610 Newport Center Drive, Suite 1400
Newport Beach, CA 92660
USA
Tel. (949) 720-1434 / Fax. (949) 720-0426
support@quests.com

Instance Monitor
Tuning Guide
Updated – May, 1999
Software Version – 1.0

Contents

Introduction

Instance Monitor is designed to monitor the overall tune of your Oracle database. This document describes:

- The Oracle architecture.
- How Instance Monitor displays that architecture.
- How you can use Instance Monitor to detect and resolve Oracle performance problems.

The aim in tuning and monitoring the Oracle instance is to ensure that data and instructions flow smoothly through and between the various processes, and that none of these flows becomes a bottleneck for the system as a whole. You can use the main **Instance Monitor** window to detect blockages or inefficiencies.

This document also contains a comprehensive glossary of Oracle and Instance Monitor terms. The glossary starts on page 51.

2

A Review of the Oracle Architecture

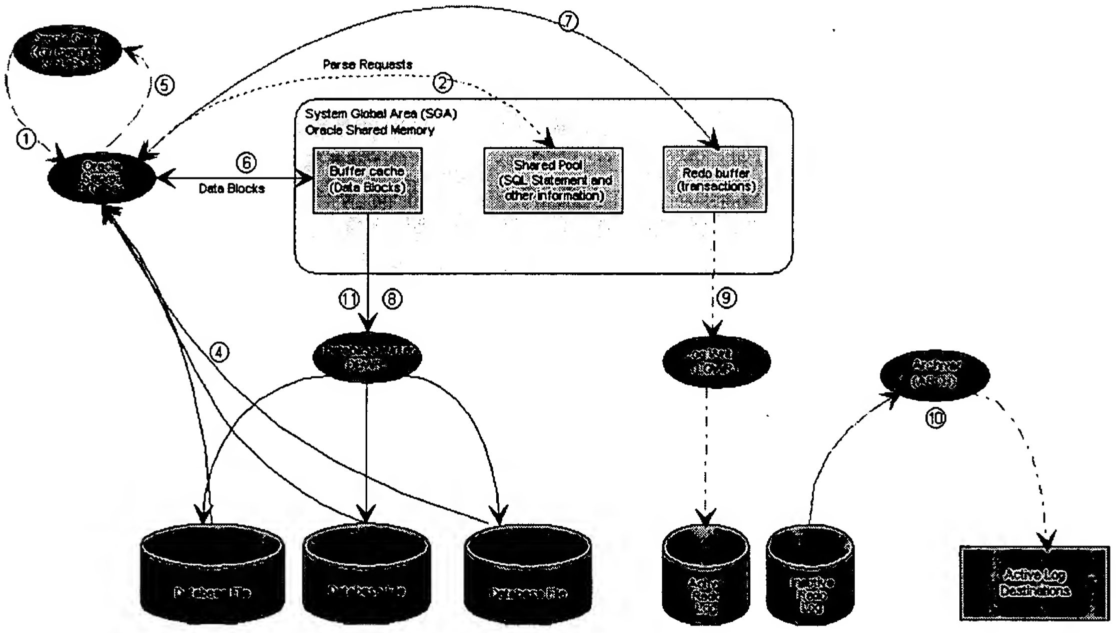
This chapter contains a summary of the architecture of Oracle databases. This architecture has been used to design the main Instance Monitor window.

By reading this chapter you will gain an understanding of the how data flows into and out of your databases.

This chapter contains the following topics:

Basic Components of an Oracle Server

The following diagram illustrates some of the basic components of an Oracle server at the memory, database, and disk layers. The numbers indicate the order of the flow of information.



Overview of the Oracle Architecture

The numbered labels in the Oracle architecture diagram correspond to the following activities:

1 The client program (for example, SQL*PLUS, Oracle Power Objects, or some other tool) sends a SELECT statement to the server process.

2 The server process looks in the shared pool for a matching SQL statement. If none is found, the server process parses the SQL and inserts the SQL statement into the shared pool.

3 The server process looks in the buffer cache for the data blocks required. If found, the data block must be moved on to the most recently-used end of the Least Recently Used (LRU) list.

4 If the block cannot be found in the buffer cache the server process must fetch it from the disk file. This requires a disk I/O.

5 The server process returns the rows retrieved to the client process. This may involve some network or communications delay.

6 When the client issues the UPDATE statement the process of parsing the SQL and retrieving the rows to be updated must occur. The update statement then changes the relevant blocks in shared memory and updates entries in the rollback segment buffers.

7 The update statement also makes an entry in the redo log buffer that records the transaction details.

8 The database-writer background process copies modified blocks from the buffer cache to the database files. The Oracle session performing the update does not have to wait for this to occur.

9 When the COMMIT statement is issued the log writer process must copy the contents of the redo log buffer to the redo log file. The COMMIT statement does not return control to the Oracle session issuing the commit until this write is complete.

10 If running in ARCHIVELOG mode, the archiver process copies full redo logs to the archive destination. A redo log is not eligible for re-use until it has been archived.

11 At regular intervals, or when a redo log switch occurs, Oracle performs a checkpoint. A checkpoint requires all modified blocks in the buffer cache to be written to disk. A redo log file cannot be re-used until the checkpoint completes.

Instance Monitor Tuning Methodology

Instance Monitor supports an approach to Oracle performance tuning that could be described as tuning by bottleneck. Instance Monitor alerts you if any component of the Oracle architecture is forming a bottleneck. Additionally, Instance Monitor provides tools that allow you to identify and rectify any inefficiency in your database's configuration.

In general, you tune your Oracle instance by reiteratively identifying bottlenecks, contention, and critical resources, and by using the advice in this chapter to remove the bottleneck or contention or improve the performance of the resource.

This chapter contains the following topics:

Summary of Tuning Steps

The following are a summary of steps that you can undertake to tune Oracle using Instance Monitor:

- Use the main **Instance Monitor** window to alert you to any obvious bottlenecks. If Instance Monitor detects that some component of Oracle constitutes a performance bottleneck, the corresponding component generates a visual or auditory alarm. The exact appearance of the alarm depends on how you have configured alarm severities. When an alarm is current you can press F1 to display help for the alarm. Clicking the object displays an-screen report. Right clicking an object displays a list of reports and help topics, and gives you access to the threshold properties for the metric.
- Use the **Activity** drilldown to display overall efficiency and resource usage. In particular, the **CPU and event waits** chart shows the amount of time database sessions are spending waiting for various resources, as well as the amount of CPU being utilized. Tuning efforts are most likely to be successful if they are concentrated on the resources being used most heavily. For more information see the *Instance Monitor User's Guide.* The **Wait activity** topic contains a discussion of the meaning and implications of various wait events.
- Other pages help you examine detail activity. For instance, the **Top Sessions** drilldown shows the users who are contributing most heavily to database activity. You can display various details for these sessions, including the SQL statements, locks, and resource usage.

Instance Monitor's approach to Oracle instance tuning is not the only valid approach. In particular, it emphasizes post-implementation tuning of Oracle applications and remedial action to correct fundamental physical configuration deficiencies in an Oracle installation.

Optimal database performance occurs when optimization is built into the application design and development process, and when the database's physical design is specifically tailored to support an accurately estimated workload. The Bibliography (see page 49) contains a list of documents that can help you do this.

Detailed tuning techniques

This section outlines detailed techniques for improving performance of Oracle databases including:

- Reducing parse overhead
- Tuning rollback segments
- Reducing lock contention
- Improving database writer performance
- Improving redo log writer performance
- Isolating and tuning problem SQL
- Relieving latch contention
- Improving database I/O
- Dealing with MTS contention
- Dealing with wait events.

Reducing parse overhead

Parsing is the process of preparing your SQL statement for execution. This process is comparable to the process a language compiler or interpreter must undertake in order to translate high-level statements into machine code. The parse process includes the following phases:

- Checking that the SQL statement is syntactically valid (that is, that the SQL conforms to the rules of the SQL language and that all keywords and operators are valid and correctly used).
- Check that the SQL is semantically valid. This means that all references to database objects (such as tables and columns) are valid.
- Check security (that is, that the user has permission to perform the specified SQL operations on the objects involved).
- Determine an execution plan for the SQL statement. The execution plan describes the series of steps that Oracle performs in order to access and update the data involved.

Parsing can be an expensive operation. Its overhead is often masked by the greater overhead of high I/O requirements. However, eliminating unnecessary parsing is always desirable.

The parse/execute ratio reflects the ratio of parse calls to execute calls. Because parsing is an expensive operation it is better to parse statements once and then execute them many times. High parse ratios (greater than 20%) can result from the following circumstances:

- If literals, rather than bind variables, are used as query parameters, the SQL must be re-parsed on every execution. You should use bind variables whenever possible, unless there is a pressing reason for using column histograms.
- Some development tools or techniques result in SQL cursors being discarded after execution. If a cursor is discarded then the parse is required before the statement can be re-executed.

If an application is discarding cursors it can be possible to relieve some of the parse overhead by creating a session cursor cache. You can do this by using the SESSION_CACHED_CURSORS configuration parameter. This setting allows Oracle to maintain a cache of SQL statements in the session memory. If a session requests a parse of a statement that it has already parsed then it might be found in the cache and re-parsing is avoided.

Tuning rollback segments

Rollback segments store original (or before image) copies of database blocks that have been changed but not committed. Rollback segments contain the information that must be restored if a ROLLBACK command is issued.

The configuration of your rollback segments can have an important effect on the performance of your database, especially for transactions that modify data. Any operation that modifies data in the database must create entries in a rollback segment. Queries that read data that has been modified by uncommitted transactions also needs to access data within the rollback segments.

Poorly tuned rollback segments can have the following consequences:

- If there are too few rollback segments, transactions may need to wait for entries in the rollback segment.
- If rollback segments are too small they may have to grow dynamically during the transaction and later shrink back (if the rollback segment has an optimal size specified).

As well as these performance-related problems, poorly tuned rollback segments can lead to transaction failure (failure to extend rollback segment) or query failure (snapshot too old).

The following guidelines serve as a starting point for rollback segment configuration for a transaction-processing environment:

- The number of rollback segments should normally be at least one quarter of the maximum number of concurrently active transactions. In batch environments, this can mean allocating a rollback segment for each concurrent job.
- Set OPTIMAL or MINEXTENTS so that the rollback segment has at least ten to twenty extents. This minimizes wastage and contention when a transaction tries to move into an already occupied extent.
- Make all extents the same size.
- Allow ample free space in the rollback segment tablespace for rollback segment expansion. Large, infrequent transactions can then extend a rollback segment when required. Use OPTIMAL to ensure that this space is reallocated when required.

It is very difficult to determine the optimal setting for rollback segments by theory alone. Rollback segments should be carefully monitored and storage adjusted as required.

Reducing lock contention

Lock, or enqueue waits occur when a session waits to obtain a lock. In most cases, this occurs because of a lock on a table or row that the session wants to lock or modify. In some circumstances, the lock involved may be an Oracle internal lock (for instance, the Space Transaction enqueue). If the database is well tuned and the application design sound, enqueue waits should be negligible. Common causes of excessive enqueue waits are:

- Contention for a specific row in the database. The application design may require that many processes update or lock the same row in the database. One common example of this is when primary keys are generated using a sequence table.
- Table locks caused by foreign keys that have not been indexed. If a non-indexed foreign key is updated then the parent table is subjected to a table lock until the transaction is complete.

- Old-style temporary tablespaces. If the tablespace nominated as the temporary tablespace has not been created with the TEMPORARY clause (introduced in Oracle 7.3), sessions may contend for the space transaction lock.

The space reserved for transactions within a data block is too small. By default, only one transaction slot for tables or two for indexes is allocated when the table or index is created. The number of transaction slots is determined by the INITRANS clause in the CREATE TABLE or INDEX statement. If additional transaction slots are required they are created, providing there is free space in the block. However, if all transaction slots are in use (and there is no free space in the block) a session that needs to lock a row in the block, encounters an enqueue wait. This occurs even if the row in question is not actually locked by another process. This can occur if both PCTFREE and INITRANS were set too low.

Improving database writer performance

Free buffer or *write complete* waits can indicate that the Database Writer process (DBWR) process is experiencing a disk I/O bottleneck.

The DBWR is the only process that writes modified database blocks from the buffer cache to the database files. The Database Writer writes asynchronously. This means that a user process never needs to wait for the Database Writer write to complete. If, however, the DBWR falls behind sufficiently, the buffer cache fills up with dirty blocks and waits occur while user processes try to introduce new blocks into the cache.

Keeping the Database Writer optimized is therefore critical to maintaining database throughput. The best way to optimize throughput is to spread I/O across multiple disk devices and allow the Database Writer to write to these disk devices in parallel. This can be achieved in two ways:

1 Multiple Database Writers can be configured using the DB_WRITERS (v7) or DBWR_IO_SLAVES configuration parameter.

2 Operating system asynchronous I/O or list I/O can be enabled. This allows the Database Writer to issue write requests against multiple disk devices simultaneously.

Experience shows that operating system asynchronous I/O performs more efficiently than multiple Database Writers. However, asynchronous I/O may not be available on all platforms, or may require special measures. In some operating systems, asynchronous I/O may require that the database be built on raw devices.

If you are configuring multiple Database Writers, you may benefit from configuring as many Database Writers as you have physical disks.

For more information see Improving database I/O on page 31.

Improving redo log writer performance

Redo logs contain details of transactions that may not yet have been written to the data files. The primary purpose of the redo logs is to allow for the recovery of the database in the event of a system or database failure.

This section covers:

- Redo log configuration
- Optimizing archiving
- Log file wait events

For information on waits see Log file space/switch waits on page 37.

Redo log configuration

When a transaction is committed, a physical write to the redo log file must occur. The write must complete before the commit call returns control to the user. Hence redo log writes can provide a limit to throughput of update intensive applications.

Redo log I/O is optimized if the log is on a dedicated device and there is no contention for the device. If this is achieved, the disk head is already in the correct position when the commit is issued and write time is minimized (the disk won't need to seek).

Because the log writes are sequential and are performed by the Logwriter processes only, there is little advantage in striping. Since LGWR is write-only to these devices, the performance degradation caused by RAID 5 is likely to be most significant, even if the volume is dedicated to redo logs (because of contention with the archiver process).

To insure against any loss of data in the event of a media failure, it is essential that the redo logs are mirrored. Oracle provides a software mirroring capability (redo log multiplexing), although hardware mirroring (RAID 1) is probably more efficient.

Because switching between redo logs results in a database checkpoint, and because a log cannot be reused until that checkpoint is completed, large and numerous logs can result in better throughput. By increasing the number of logs, you reduce the possibility that a log is required for reuse before its checkpoint is complete. By increasing the size of the logs you reduce the number of checkpoints that must occur.

The optimal size for your redo logs depends on your transaction rate. You should size the logs so that switches do not occur too rapidly. Since you usually allocate dedicated devices for redo logs, there is likely to be substantial disk capacity available for logs. It is often easiest to over-configure the log size and number initially. Log sizes of 64 to 256 megabytes are not uncommon. Configuring from 10 to 20 redo logs is also not unusual.

Optimizing archiving

Archived logs are copies of on-line redo logs. They can be used to recover a database to point of failure or to another point in time after a backup has been restored. Archive logging is also required if on-line backups are desired.

Once a redo log file is filled Oracle moves to the next log file. The archiver process (ARCH) copies the recently-filled log to an alternate location. If the archiver reads from a log on the same physical device as the current log being written, the sequential writes of the log writer are disrupted. If the log writer falls sufficiently behind, the database can stall (since a log file cannot be reused until it has been archived).

Therefore, it is important to optimize the performance of the archiver. Contention between the archiver and the log writer can be minimized by alternating redo logs over two devices. The redo log writer can then write to one device while the archiver is reading from the other device. Since the archiver must be capable of writing at least as fast as the log writer, the archive destination should either be a dedicated device or a dedicated set of disks in a RAID 0+1 (mirrored and striped) configuration.

Log file wait events

Just as Oracle sessions must wait for database file I/O, they must also wait for log file I/O. Such waits occur whenever a COMMIT statement causes a write to the redo log. The session issuing the COMMIT waits on the *log file sync* event. When the log writer issues the I/O request, it waits on the *log file parallel write* event.

Both these wait events are inevitable and often account for between 10-20% of total non-idle wait times in a healthy database.

The average wait-time for a log file parallel write is an important measure. It indicates how quickly the log writer process can flush the redo buffer. It is a good indicator of the efficiency of the redo log device. Values under one hundredth of a second are good, and values of up to five hundredths of a second are not unusual. Values above this range may indicate contention for the redo log device.

Isolating and tuning problem SQL

The **Top SQL** screens allow you to identify the SQL that is consuming the most resources on your system. Tuning this SQL can be one of the most effective ways of tuning your database.

See Using the Top SQL drilldown (in the *Instance Monitor User's Guide*) for details of the **Top SQL** drilldown.

See Using Explain Plan on page 41 for information on interpreting SQL execution plans

This section contains guidelines on SQL tuning. These include:

- Indexing (see below)
- Taking advantage of the Cost Based Optimizer (page 17)
- Avoid accidental table scans (page 18)
- Optimize necessary table scans (page 19)
- Use array processing (page 22)
- New SQL tuning facilities in 7.3 and Oracle 8 (page 22)
- Optimizer hints (page 23).

Indexing

Indexes exist primarily to improve the performance of SQL statements. In many cases, establishing good indexes is the best path to optimal performance.

Use concatenated indexes

Try not to use two indexes when one would do. If searching for SURNAME and FIRSTNAME, don't necessarily create separate indexes for each column. Instead create a concatenated index on both SURNAME and FIRSTNAME. You can use the leading portion of a concatenated index on its own. If you sometimes query on the SURNAME column without supplying the FIRSTNAME then SURNAME should come first in the index.

Over index to avoid a table lookup

Sometimes you can improve SQL execution by over indexing. Over indexing involves concatenating columns that appear in the SELECT clause, but not in the WHERE clause to the index.

For example, if you are searching on SURNAME and FIRSTNAME in order to find EMPLOYEE_ID the concatenated index on SURNAME and FIRSTNAME allows you to quickly locate the row containing the appropriate EMPLOYEE_ID. However, you need to access both the index and the table. If there is an index on SURNAME, FIRSTNAME, and EMPLOYEE_ID, the query can be satisfied using the index alone. This technique can be particularly useful when optimizing joins, since intermediate tables in a join are sometimes queried merely to obtain the join key for the next table.

Consider advanced indexing options

Oracle default B*-tree indexes are flexible and efficient. They are suitable for the majority of situations. However, Oracle offers a number of alternate indexing schemes that can improve performance in specific situations. These include:

- **Index clusters** allow rows from one or more tables to be located in cluster key order. Clustering tables can result in a substantial improvement in join performance. However, table scans of individual tables in the cluster can be severely degraded. Index clusters are usually only recommended for tables that are always accessed together. Even then, alternatives (such as de-normalization) should be considered.
- In **hash clusters**, the key values are translated mathematically to a hash value. Rows are stored in the hash cluster based on this hash value. Locating a row when the hash key is known may require only a single I/O, rather than the two or three I/Os required by an index lookup. However, range scans of the hash key cannot be performed. If the cluster is poorly configured hash key retrieval can degrade. If the size of the cluster changes, then overflows on the hash keys can occur or the cluster can become sparsely populated. In this case table scans are less efficient.
- **Bit-mapped indexes** were introduced in Oracle 7.3. They suit queries on multiple columns that have a few distinct values. They are more compact than a concatenated index. Unlike the concatenated index, they can support queries in which the columns appear in any combination. However, bit-mapped indexes are not suitable for tables that have high modification rates, since locking of bit-mapped indexes occurs at the block, rather than row level. They are also not suitable for columns with large numbers of distinct values.

Make sure your query uses the best index

Novice SQL programmers are often satisfied if the execution plan for their SQL statement uses any index. However, there is sometimes a choice of indexed retrievals and the Oracle optimizer (especially the older Rule-based optimizer) does not always choose the best index. Make sure that the indexes being selected by Oracle are the most appropriate and use hints to change the index if necessary.

Taking advantage of the Cost Based Optimizer

The component of the Oracle software that determines the execution plan for an SQL statement is known as the optimizer. Oracle supports two approaches to query optimization. They are:

- The rule-based optimizer determines the execution plan based on a set of rules. The rules rank various access paths. For example, an index-based retrieval has a lower rank than a full table scan. A rule-based optimizer uses indexes wherever possible.
- The cost-based optimizer determines the execution plan based on an estimate of the computer resources (the cost) required to satisfy various access methods. The cost-based optimizer uses statistics (including the number of rows in a table and the number of distinct values in indexes) to determine the optimum plan.

Early experiences with the cost-based optimizer in Oracle 7.0 and 7.1 were often disappointing and gave the cost-based optimizer a bad reputation. However, the cost-based optimizer has been improving in each release. The rule-based optimizer is virtually unchanged since Oracle 7.0. Advanced SQL access methods (such as star and hash joins) are only available when you use the cost-based optimizer.

The cost-based optimizer is the best choice for almost all new projects. Converting from rule to cost-based optimization is worthwhile for many existing projects. Consider the following guidelines for getting the most from the cost-based optimizer:

- **OPTIMIZER_MODE**. The default mode of the cost-based optimizer (optimizer_mode=CHOOSE) attempts to optimize the throughput (that is, the time taken to retrieve all rows) of SQL statements. It often favors full table scans over index lookups. When converting to cost-based optimization, many users are disappointed to find that previously well tuned index lookups change to long running table scans. To avoid this, set OPTIMIZER_MODE=FIRST_ROWS in init.ora or ALTER SESSION SET OPTIMIZER_GOAL=FIRST_ROWS in your code. This instructs the cost-based optimizer to minimize the time taken to retrieve the first row in your result set and encourages the use of indexes.

- **Hints**. No matter how sophisticated the cost-based optimizer becomes, there are still occasions when you need to modify its execution plan. SQL hints are usually the best way of doing this. Using hints, you can instruct the optimizer to pursue your preferred access paths (such as a preferred index), use the parallel query option, select a join order and so on. Hints are entered as comments following the first word in an SQL statement. The plus sign (+) in the comment lets Oracle know that the comment contains a hint. Hints are fully documented in the Oracle Server Tuning guide and some of the most popular hints are summarized in Optimizer Hints on page 23. In the following example, the hint instructs the optimizer to use the CUST_I2 index:

```
SELECT /*+ INDEX(CUSTOMERS CUST_I2) */ *
  FROM CUSTOMERS
 WHERE NAME=:cust_name
```

- **Analyze your tables**. The cost-based optimizer's execution plans are calculated using table statistics collected by the ANALYZE command. Make sure you analyze your tables regularly, that you analyze all your tables, and that you analyze them at peak volumes (for instance, don't analyze a table just before it is about to be loaded by a batch job). For small to medium tables, use ANALYZE TABLE *TABLE_NAME* COMPUTE STATISTICS. For larger tables take a sample (for example, ANALYZE TABLE *TABLE_NAME* ESTIMATE STATISTICS SAMPLE 20 PERCENT).
- **Use histograms**. Prior to Oracle 7.3, the cost-based optimizer included the number of distinct values in a column but not the distribution of data within the column. This meant that it might decline to use an index on a column with only a few values, even if the particular value in question was very rare and would benefit from an index lookup. Histograms, introduced in Oracle 7.3, allow column distribution data to be collected and allow the cost-based optimizer to make better decisions. You can create histograms with the FOR COLUMNS clause of the ANALYZE command (for instance ANALYZE TABLE *TABLE_NAME* COMPUTE STATISTICS FOR ALL INDEXED COLUMNS). You cannot take advantage of histograms if you are using bind variables.

Avoid accidental table scans

One of the most fundamental SQL tuning problems is the accidental table scan. Accidental table scans usually occur when the SQL programmer tries to perform a search on an indexed column that can't be supported by an index. This can occur when:

- **Using != (not equals to)**. Even if the **not equals** condition satisfies only a small number of rows, Oracle does not use an index to satisfy such a condition. Often, you can re-code these queries using > or IN conditions, which can be supported by index lookups.
- **Searching for NULLS**. Oracle won't use an index to find null values, since null values are not usually stored in an index (the exception is a concatenated index entry where only some of the values are NULL). If you're planning to search for values that are logically missing, consider changing the column to NOT NULL with a DEFAULT clause. For example, you could set a default value of UNKNOWN and use the index to find these values. Interestingly, recent versions of Oracle can index to find values that are NOT NULL - if the cost-based optimizer determines that such an approach is cost-effective.
- **Using functions on indexed columns**. Any function or operation on an indexed column prevents Oracle from using an index on that column. For instance, Oracle can't use an index to find SUBSTR(SURNAME,1,4)='SMIT'. Instead of manipulating the column, try to manipulate the search condition. In the previous example, a better formulation would be SURNAME LIKE 'SMIT%'.

Optimize necessary table scans

In many cases, avoiding a full table scan by using the best of all possible indexes is your aim. Often though, a full table scan cannot be avoided. In these situations, consider some of the following techniques to improve table scan performance:

- Parallel query option
- Reduce the size of the table
- Use the CACHE hint
- Use partitioning
- Fast full index scans

Using the parallel query option

Oracle's Parallel Query Option is the most effective (and most resource intensive) way of improving the performance of full table scans. Parallel Query allocates multiple processes to an SQL statement that is based (at least partially) on a full table scan. The table is partitioned into distinct sets of blocks and each process works on a different set of data. Further processes may be allocated, or the original processes recycled, to perform joins, sorts, and other operations.

The approach of allocating multiple processes to the table scan can reduce execution time dramatically if the hardware and database layout is suitable. In particular, the host computer should have multiple CPUs or the database should be spread across more than one disk device.

You can enable the Parallel Query option with a PARALLEL hint or make it the default for a table with the PARALLEL table clause.

Reducing the size of the table

The performance of a full table scan is generally proportional to the size of the table to be scanned. There are ways of reducing the size of the table quite substantially and thereby improving full table scan performance. These include:

- **Reduce PCTFREE**. The PCTFREE table setting reserves a certain percentage of space in each block to allow for updates which increase the length of a row. By default, PCTFREE is set to 10%. If your table is rarely updated, or if the updates rarely increase the length of the row, you can reduce PCTFREE and hence reduce the overall size of the table.
- **Increase PCTUSED**. The PCTUSED table setting determines at what point blocks that have previously hit PCTFREE becomes eligible for inserts again. The default value is 40%. This means that after hitting PCTFREE, the block only becomes eligible for new rows when deletes reduce the amount of used space to 40%. If you increase PCTREE, rows are inserted into the table at an earlier time, blocks are fuller on average, and the table is smaller. There may be a negative effect on INSERT performance. You must assess the trade off between scan and insert performance.
- **Relocate LONG columns**. If you have LONG (or big VARCHAR2) columns in the table that are not frequently accessed and never accessed via a full table scan (perhaps a bitmap image or embedded document) you should consider relocating these to a separate table. By relocating these columns you can substantially reduce the table's size and hence improve full table scan performance. Oracle8 large objects (LOBs) over 4K in length are always automatically stored in a separate segment.

Using the CACHE hint

Normally, rows retrieved by most full table scans are flushed almost immediately from Oracle's cache. This is sensible, otherwise full table scans could completely saturate the cache and push out rows retrieved from index retrievals. However, this does mean that subsequent table scans of the same table are unlikely to find a match in the cache and therefore incur a high physical I/O rate.

You can encourage Oracle to keep these rows within the cache by using the CACHE hint or the CACHE table setting. Oracle then places the rows retrieved at the Least Recently Used end of the LRU chain and they persist in the cache for a much longer period of time.

Using partitioning

If the number of rows you want to retrieve from a table is greater an index lookup could effectively retrieve, but still only a fraction of the table itself (say between 10 and 40% of total), you could consider partitioning the table.

For instance, suppose that a SALES table contains all sales records for the past 4 years and you frequently need to scan all sales records for the current financial year in order to calculate year-to-date totals. The proportion or rows scanned is far greater than an index lookup would comfortably support but is still only a fraction of the total table.

If you partition the table by financial year you can restrict processing to only those records which match the appropriate financial year. This could potentially reduce scan time by 75% or more.

In Oracle 7.3 you can create separate tables for each financial year and then create a partition view. A partition view is a UNION ALL view with check constraints on each table. These enforce the partitioning. Scans on the view that specify a particular financial year clause only need to scan the appropriate table.

In Oracle 8, true partitioned tables can be created. In an Oracle 8 table, partitioned by financial year, rows for the appropriate financial year would be stored in distinct partitions and the optimizer would restrict queries against a particular financial year to the appropriate partition.

Using the fast full index scan

If a query needs to access all or most of the rows in a table, but only a subset of the columns, you can consider using a fast full index scan to retrieve the rows. To do this, you need to create an index that contains all the columns included in the SELECT and WHERE clauses of the query. If these columns comprise only a small subset of the entire table, the index is substantially smaller. Oracle is able to scan the index more quickly than it could scan the table. There is an overhead involved in maintaining the index that affects the performance of INSERT, UPDATE, and DELETE statements.

Using an index to perform the equivalent of a full table scan is possible in earlier versions of Oracle, but only since Oracle 7.3 is the index scan able to use multi-block read and parallel query capabilities.

Use array processing

Array processing refers to Oracle's ability to insert or select more than one row in a single operation. For SQL, which deals with multiple rows of data, array processing usually results in reductions of 50% or more in execution time (more if you're working across the network). In some application environments, array processing is implemented automatically and you won't have to do anything to enable this feature. In other environments, array processing must be totally implemented by the programmer.

Many programmers implement huge arrays. This can be excessive and may even reduce performance by increasing memory requirements for the program. Most of the gains of array processing are gained by increasing the array size from 1 to about 20. Further increases result in diminishing gains. You do not normally see much improvement when increasing the array size over 100.

New SQL tuning facilities in 7.3 and Oracle 8.0

Each release of Oracle introduces new and improved SQL performance features. In particular the cost-based optimizer contains improvements in each release - many of which are undocumented. Some of the newer Oracle features that can help your SQL performance are:

- **Hash joins**. This new join algorithm improves the performance of joins that previously used the sort-merge algorithm. Introduced in 7.3 and invoked automatically unless HASH_JOIN_ENABLED=FALSE.
- **Anti-joins**. The anti-join algorithm allows efficient execution of queries that use NOT IN sub-queries. These types of queries were typically performance problems in earlier versions of Oracle. You can invoke the anti-join with the MERGE_AJ or HASH_AJ hints (in the sub-query) or by setting ALWAYS_ANTI_JOIN=TRUE.
- **Histograms**. Histograms allow the cost-based optimizer to make more informed decisions regarding the distribution of data within a column. Introduced in 7.3 and created using the FOR COLUMNS clause of the ANALYZE command.
- **Partitioning**. The partition view introduced in 7.3 and the partitioned table introduced in 8.0 allow subsets of large tables to be processed separately.
- **Parallel DML**. Starting with Oracle 8.0, DML statements (UPDATE, INSERT, DELETE) can be processed using parallel processing. For DELETE and UPDATE operations, the table involved should be partitioned.

- **Fast full index scan**. Oracle 7.3 can perform fast index scans using multi-block reads and parallel query processing if the index includes all the columns required to satisfy the query.

Optimizer Hints

Optimizer hints appear as a comment following the first word of the SQL statement (for example, SELECT, INSERT, DELETE, or UPDATE). Hints are differentiated from other comments by the presence of the plus sign (+) following the opening comment delimiter (/*). For instance, the FULL hint in the following example tells the optimizer to perform a full table scan when resolving the query:

```
SELECT /*+ FULL(E) */ *
FROM employee e
WHERE salary > 1000000
```

The following table shows the hints that can be used:

ALL_ROWS

Use the cost-based optimizer and optimize for the retrieval of all rows.

AND_EQUAL(table_name index_name index_name ....)

Retrieve rows from the specified table using each of the specified indexes and merge the results.

APPEND

Invokes a direct load insert. Only valid for INSERT ... SELECT FROM statements.

BITMAP(table_name index_name)

Retrieve rows from the specified table using the specified bitmap index.

CACHE(table_name)

Encourages rows retrieved by a full table scan to remain in the buffer cache of the SGA.

CHOOSE

If statistics have been collected for any table involved in the SQL statement, use cost-based or all-rows optimization, otherwise use rule-based optimization.

CLUSTER(table_name)

Use a cluster scan to retrieve table rows.

DRIVING_SITE(table_name)

For a distributed SQL statement, causes the site at which the specified table resides to be the driving site.

FIRST_ROWS

Specifies that the cost-based optimizer should optimize the statement to reduce the cost of retrieving the first row only.

FULL(table_name)

Use a full table scan to retrieve rows from the specified table.

HASH(table_name)

Use a hash scan to retrieve rows from the specified table. The table must be stored in a hash cluster.

HASH_AJ

Perform an anti-join using hash join methodology. This hint must appear after the select statement, not in sub-query.

HASH_SJ

Appears within an EXISTS sub-query. Invokes a hash semi-join.

INDEX(table_name [index_name])

Use the specified index to retrieve rows from the table or, if no index specified, use any index.

INDEX_ASC(table_name [index_name])

Specifies an ascending index range scan using the specified index or, if no index is specified, any suitable index.

INDEX_COMBINE(table_name [index_name...])

Instructs the optimizer to combine the specified bitmap indexes. If no bitmap indexes are specified, the optimizer chooses suitable bitmap indexes.

INDEX_DESC(table_name [index_name])

Specifies a descending index range scan using the specified index or, if no index is specified, any suitable index.

INDEX_FFS(table_name [index_name])

Invokes a fast full index scan using the specified index or, if no index is specified, any suitable index. A fast full scan reads all the index in block order using multi-block reads and possibly parallel query.

MERGE

Instructs the optimizer to perform complex view merging when resolving a query based on a view or that includes a sub-query in the where clause.

NO_MERGE

Instructs the optimizer not to perform complex view merging when resolving a query based on a view or that includes a sub-query in the where clause.

MERGE_AJ

Perform an anti-join using sort-merge join method. This hint must appear after the select statement, not in a sub-query

MERGE_SJ

Appears within an EXISTS sub-query. Invokes a sort-merge semi-join.

NO_EXPAND(table_name)

Oracle sometimes expands statements with OR conditions into multiple SQL statements combined by a union operation. This hint instructs the optimizer not to do this, even if it calculates that such a transformation would be beneficial.

NO_INDEX(table_name [index_name])

No index suppresses the use of the named indexes or, if no indexes are specified, all indexes on the nominated table.

NO_PUSH_PRED

Do not push join conditions from the where clause into a view or sub-query.

NOAPPEND

Suppresses direct load insert in an INSERT... SELECT FROM... statement.

NOCACHE(table_name)

Discourages Oracle from keeping rows retrieved by a full table scan in the buffer cache of the SGA. Overrides the cache setting on the CREATE or ALTER TABLE statement.

NOPARALLEL(table_name)

Don't use parallel processing for the SQL statement. Overrides the parallel setting on the create or alter table statement

NOPARALLEL_INDEX(table_name index_name)

Suppresses parallelism in fast full index scans or in partitioned index access.

NOREWRITE

(Oracle 8i)

Prevents the SQL statement from being rewritten to take advantage of materialized views. It overrides the server parameter QUERY_REWRITE_ENABLED.

ORDERED

Instructs the optimizer to join the tables in exactly the left to right order specified in the FROM clause.

ORDERED_PREDICATES

(Oracle 8i)

Causes predicates in the WHERE clause to be evaluated in the order in which they appear in the WHERE clause.

PARALLEL(table_name , degree_of_parallelism)

Instructs the optimizer to perform parallel scans on the nominated table. If no degree of parallelism is specified, the default is used.

PARALLEL_INDEX(table_name [index_name])

Parallelizes a fast full index scan or an index scan against a partitioned index.

PQ_DISTRIBUTE(table_name outer_distribution inner_distribution)

This query determines how a parallel join using TABLE_NAME is executed. Valid options for OUTER_DISTRIBUTION and INNER_DISTRIBUTION are (not all combinations are valid) HASH, BROADCAST, NONE, PARTITION.

PUSH_JOIN_PRED/PUSH_PRED

Push join conditions from the where clause into a view or sub-query.

PUSH_SUBQ

This hint causes sub-queries to be processed earlier in the execution plan. Normally sub-queries are processed last unless the SQL statement is transformed into join.

REWRITE(view_name [view_name...])

(Oracle 8i)

Restrict query rewrite to only those materialized views specified in the hint.

ROWID(table_name)

Perform a ROWID access.

RULE

Use rule based optimization.

STAR

Consider the STAR join methodology in preference to other methods.

STAR_TRANSFORMATION

(Oracle 8.0+)

Requests that the star transformation optimization be performed. This transforms a star query into a alternate form which can take advantage of bitmap indexes.

USE_CONCAT

Oracle sometimes expands statements with OR conditions into multiple SQL statements combined by union all. This hint instructs the optimizer to do this, even if it calculates that such a transformation would not be beneficial.

USE_HASH(table_name)

When joining to this table, use the hash join method.

USE_MERGE(table_name)

When joining to this table, use the sort-merge join method.

USE_NL(table_name)

When joining to this table, use the nested-loops join method.

Relieving latch contention

Operations that affect the contents of the SGA require that a process acquire a latch. A latch is similar to a lock, but instead of preventing two sessions from concurrently changing the same row, a latch prevents two sessions from altering the same area in shared memory at once.

Latches are usually held for a very brief interval. In a healthy database there should be little or no contention for latches. Unfortunately, very busy databases often suffer considerably from latch contention.

If a process requires a latch and cannot obtain it on the first attempt, a latch miss results. The session repeatedly attempts to obtain the latch up to value of the configuration parameter SPIN_COUNT. This technique is known as acquiring a spin lock. If the session still cannot obtain the latch then the session relinquishes the CPU and a latch sleep results. A latch sleep is recorded as a latch free wait. When the session wakes up it repeats the attempt to obtain the latch.

The latches that contribute to a high proportion of misses or sleeps deserve attention. Not surprisingly, the latches that are used most heavily (and which typically suffer the most contention) are the latches associated with the three major areas of the SGA. They are:

1. Buffer cache latches
2. Library cache latches
3. Redo buffer latches.

For more information on latch contention, see Spin count and latch sleeps on page 30.

Buffer cache latches

Two main latches protect data blocks in the buffer cache. The cache buffer LRU chain latch must be obtained in order to introduce a new block into the buffer cache and when writing a buffer back to disk.

A cache buffer chains latch is acquired whenever a block in the buffer cache is accessed (pinned).

Contention for these latches usually typifies a database that has very high I/O rates. It is possible to reduce contention for the cache buffer LRU chain latch by increasing the size of the buffer cache. This reduces the rate at which new blocks are introduced into the buffer cache.

Reducing contention for the cache buffer chains latch usually requires the reduction of logical I/O rates (by tuning and minimizing the I/O requirements of application SQL).

You can create additional cache buffer LRU chain latches by adjusting the configuration parameter DB_BLOCK_LRU_LATCHES. You can reduce load on the cache buffer chain latches by increasing the configuration parameter _DB_BLOCK_HASH_BUCKETS.

Library cache latches

The library cache latches protect the cached SQL statements and object definitions held in the library cache within the shared pool.

The library cache latch must be obtained in order to add a new statement to the library cache. During a parse request, Oracle searches the library cache for a matching statement. If one is not found, Oracle parses the SQL statement, acquires the library cache latch, and inserts the new SQL. Contention for the library cache latch can occur when an application generates very high quantities of unique, non-sharable SQL (usually because literals have been used instead of bind variables).

If the library cache latch is a bottleneck, try to improve the use of bind variables within your application. Misses on this latch can also be a sign that your application is parsing SQL at a high rate and may be suffering from excessive parse CPU overhead as well.

The library cache pin latch must be obtained when a statement in the library cache is re-executed. Misses on this latch occur when there are very high rates of SQL execution. There is little you can do to reduce the load on this latch, although using private rather than public synonyms (or even direct object references such as OWNER.TABLE) may help.

The _KGL_LATCH_COUNT parameter controls the number of library cache latches. The default value should be adequate but you may want to increase it if contention for the library cache latch cannot be resolved.

Redo buffer latches

Two latches control access to the redo buffer. The redo allocation latch must be acquired in order to allocate space within the buffer. If the redo log entry to be made is greater than the configuration parameter LOG_ENTRY_MAX_SIZE, the session that acquires the redo allocation latch can copy the entry into the redo buffer immediately while holding the allocation latch.

If the log entry is greater than LOG_ENTRY_MAX_SIZE, the session releases the redo allocation latch and acquires the redo copy latch in order to copy the entry.

There is only one redo allocation latch, but there may be up to LOG_SIMULTANEOUS_COPIES allocation latches.

If you see contention for the redo allocation latch, reduce the number of redo buffer copies that occur on this latch by decreasing LOG_ENTRY_MAX_SIZE. If you see contention for the redo copy latch, increase the number of copy latches by increasing the value of LOG_SIMULTANEOUS_COPIES.

Spin count and latch sleeps

If a session sleeps (because it cannot obtain a latch) response time is significantly degraded. You can decrease the probability of the session sleeping by increasing the value of the configuration parameters _SPIN_COUNT (also called _LATCH_SPIN_COUNT). This parameter controls the number of attempts the session makes to obtain the latch before sleeping.

Spinning on the latch consumes CPU. If you increase this parameter you may see an increase in your systems overall CPU utilization. If your computer is near 100% CPU utilization and your application is throughput rather than response time driven, you could consider decreasing _SPIN_COUNT in order to conserve CPU.

Adjusting _SPIN_COUNT is a trial and error process. In general, only increase this parameter if there is ample free CPU resources available on your system and decrease it only if there is no spare CPU capacity.

Improving database I/O

A number of performance indicators can suggest a need to improve Oracle disk I/O.

A major aim of configuring an Oracle server is to ensure that disk I/O does not become a bottleneck. While there may be some differences in the performance of disk devices from various vendors (especially if the devices are in some sort of RAID configuration) the major restraint on disk I/O is the number of disks acquired and the spread of I/O between these devices.

If possible, estimate the physical I/O that is generated by your database, and use this figure to determine the number of devices that would support the configuration.

For more information see:

- Disks for redo devices
- Redundant Array of Inexpensive Disks (RAID)
- Striping.

Disks for redo devices

When a transaction is committed, the redo log entry in the redo log buffer must be written to disk. The characteristics of the redo log writes are very different from that of the data file I/O. The differences include:

- They are sequential I/Os. Each access follows the previous access on the disk. The disk drive does not have to seek for the disk block to access. This means sequential I/Os are much faster than random I/Os. Most disk devices can perform about one hundred sequential I/Os per second.
- The I/Os are write-only and attempt to write through any disk cache.

These factors combined suggest that transaction processing is optimized if a redo log is on a dedicated disk device. Of course, if your database is primarily read-only, redo log I/O is unlikely to be an issue and the redo logs can be placed in (virtually) any convenient location.

Redundant Array of Inexpensive Disks (RAID)

A Redundant Array of Inexpensive Disks (RAID) array is an increasingly popular way of delivering fault tolerant, high-performance disk configurations. There are a number of levels of RAID and a number of factors to take into consideration when deciding on a RAID configuration, and the level of RAID to implement.

There are three levels of raid commonly provided by storage vendors. They are:

RAID 0 Sometimes referred to as striping disks. In this configuration, a logical disk is constructed from multiple physical disks. The data contained on the logical disk is spread evenly across the physical disk, hence random I/Os are also likely to be spread evenly. There is no redundancy built into this configuration, so if a disk fails it must be recovered from backup.

RAID 1 Referred to as disk mirroring. In this configuration, a logical disk is comprised of two physical disks. In the event that one physical disk fails, processing can continue using the other physical disk. Each disk contains identical data and writes are processed in parallel so there should be no negative effects on write performance. Two disks are available for reads, so there can be an improvement in read performance.

RAID 5 A logical disk is comprised of multiple physical disks. Data is arranged across the physical devices in a similar way to disk striping (RAID 0). However a certain proportion of the data on the physical devices is parity data. This parity data contains enough information to derive data on other disks should a single physical device fail.

It's common to combine RAID 0 and RAID 1. Such striped and mirrored configurations offer protection against hardware failure together with spread of I/O load.

For more information see Performance implications of RAID.

Performance implications of RAID

Both RAID 0 and RAID 5 improve the performance of concurrent random reads by spreading the load across multiple devices. RAID 5, however, tends to degrade write I/O, because both the source block and parity block must be read and then updated.

Neither RAID 0 or RAID 5 offer any performance advantages over single disk configurations when sequential reads or writes are being undertaken.

The performance of a combined RAID 0 and RAID 1 for database files, and RAID 1 for redo logs is generally superior to any other configuration. It also offers full protection from media failure. RAID5, however, requires less disk space than a RAID 0+1 configuration, and may provide acceptable performance in many circumstances.

Striping

The ultimate limit on I/O performance is dictated by the number of devices available and the spread of data across these devices. You should ensure that there are a sufficient number of disks to support your projected I/O rates. You should also ensure that data is spread as evenly as possible across these disks, and that there are no disk hot-spots.

There are three ways to spread data across devices. They are:

1 RAID 0 or striping.

2 RAID 5

3 Oracle striping

RAID 5 can decrease write performance unless the raid array is associated with a battery-backed memory cache (and quite often even then).

Generally, RAID 0 is recommended on performance grounds. If RAID 0 is not available, you should manually stripe your tablespaces across multiple devices. Manual (or Oracle) striping is achieved by allocating many small files to each tablespace and spreading these files across the multiple disks.

Because a table extent must be located within a single database file, tables consisting of a single extent cannot be manually striped. In this case, you must ensure that heavily utilized tables (and indexes) are composed of a large number of extents. You may decide to reduce the size of your data files to that of a single extent (plus a one block overhead).

Dealing with MTS contention

When multi-threaded servers are implemented, multiple Oracle clients share a smaller number of server processes. This can save memory and in some cases improve performance. However, if the number of shared servers is too small, then client sessions may queue for an available server. Response time and throughput might suffer dramatically.

A similar problem can occur if there are insufficient dispatchers.

If Instance Monitor detects a bottleneck in the Shared servers or dispatchers, then an alarm (such as the Multi-threaded server alarm) becomes current on the appropriate component within the Server Processes panel. If the alarms persist, you should alter the setting for MTS_MAX_DISPATCHERS or MTS_MAX_SERVERS to ensure that sufficient processes are available for your workload. You could also consider using dedicated servers.

You can view information about MTS activity on the Server activity tab.

Dealing with wait events

Whenever an Oracle session is not actually consuming or waiting for CPU resources, it will usually be in one of a number of wait events. For instance, a session may be waiting for some I/O request to be performed, for free space in the SGA, for network traffic or for an internal Oracle resource such as a latch. Some waits, such as those for datafile or log file I/O are normal and unavoidable (although you may be able to reduce their magnitude). Other waits, such as those for latches or buffers in the SGA may indicate inefficiency or bottlenecks.

In a perfect Oracle implementation, the Oracle server process is able to perform its tasks using its own resources without experiencing any delays. However, in reality Oracle sessions often wait on system or database requests or for resources to become available.

During a typical transaction, the Oracle session may need to wait for various resources to become available:

1. While the application is idle, the server process is waiting for a message from the client.
2. When the server process parses a new SQL statement, and the statement has not previously been executed, it has to acquire a latch to add the new statement to the library cache. If the latch required is held by another session, the server process may have to wait for the latch to become available.
3. The server process also has to acquire a latch when executing a SQL statement held in the shared pool. It may have to wait on the latch if it is currently held by a different session.
4. When accessing a data block in the buffer cache, the server process has to change the location of the block on the least recently used (LRU) list. This requires obtaining and possibly waiting for the appropriate latch.
5. If the block is not in the buffer cache, the session has to issue and wait for an I/O request to obtain the block. Moving a new block into the buffer cache also requires a latch that might be unavailable and cause a wait.
6. Changing the data block requires obtaining a latch both to change the block itself and to make an entry in the redo log buffer. Additionally, if there is insufficient free space in the redo log buffer, the session needs to wait for the Logwriter process to make space available.
7. When a COMMIT is issued, the session must wait for the Logwriter process to write the blocks in question to the redo log file.

8 The Logwriter session itself may need to wait if the redo log is full and the next redo log has an outstanding checkpoint or archive operation outstanding.

There are many reasons why an Oracle session may need to wait. Some of these waits (such as waiting for I/O operations) are inevitable. However, you can reduce them in many cases by tuning I/O, the buffer cache, or the SQL involved. Other operations (such as waiting for latches) may indicate inefficiencies in your configuration and opportunities for further tuning.

For more information about different types of waits see:

- DB file waits (see below)
- Log file sync/write waits (page 37)
- Log file space/switch waits (page 37)
- Buffer busy waits (page 38)
- Enqueue waits (page 38)
- Free buffer waits (page 39)
- Write complete waits (page 39)
- Latch free waits (page 40).

DB file waits

Wait conditions starting with the phrase DB FILE (for example, DB FILE PARALLEL WRITE, DB FILE SCATTERED READ, DB FILE SEQUENTIAL READ, DB FILE SINGLE WRITE) all occur when an Oracle session issues an I/O request against an Oracle datafile. The session uses the operating system's read system call and wait while the I/O subsystem performs the I/O.

As noted earlier, database-file writes are only performed by the Database Writer. The db file write waits are never experienced by user sessions. However, user sessions do read data from database files directly and almost always experience db file read waits.

Unless your entire database is cached in memory, waiting for db file I/O is inevitable. The presence of db file waits does not indicate that anything is wrong within the database. In most healthy databases, db file waits account for about 80-90% of all non-idle wait times.

DB file waits can be reduced by:

- Optimizing disk I/O, striping datafiles (see Improving database I/O on page 31).

- Reducing I/O requirements by increasing the size of the buffer cache.
- Reducing the I/O requirements of SQL statements through SQL tuning (see Isolating and tuning problem SQL on page 15).

Log file sync/write waits

Just as Oracle sessions must inevitably wait for db file I/O, they must also wait for log file I/O. These waits occur when a COMMIT is issued. A COMMIT causes the Redo log writer session to flush the contents of the redo log to the redo file. The user session must wait for this write operation to complete before the commit statement returns control.

The session issuing the COMMIT waits on the log file sync event. When the Log Writer process issues the I/O request, it waits on the log file parallel write event.

Both wait events are inevitable and usually account for between 10-20% of total non-idle wait times in a healthy database.

The average wait time for a log file parallel write is an important measure. It indicates how quickly the log writer process can flush the redo buffer. It is a good indicator of the efficiency of the redo log device. Values of 0.2 hundredths of a second are good, and values of up to 5 hundredths of a second are not unusual. Values above this range may indicate contention for the redo log device. For more information see Improving redo log writer performance on page 13.

Log file space/switch waits

Log file space/switch waits occur when a redo log entry cannot be made because:

- There is no free space in the redo log buffer.
- A redo log cannot be written to because it is in the process of being switched.

The following error messages may also be included in the alarm log for the database instance:

- **Cannot allocate new log...checkpoint not complete**
- **Cannot allocate new log...**
- **All online logs need archiving.**

The incidence of these events should be negligible in a well-tuned database. Prior to Oracle 7.3, these two conditions are combined in the event log file space/switch. From 7.3 onwards, they are defined by the following events:

- Log buffer space. Waiting for free space in the redo log buffer. You can reduce this wait by increasing the size of the log buffer (LOG_BUFFER parameter) or by optimizing the performance of the log writer.
- Log file switch (checkpoint incomplete). The next redo log cannot be used because a checkpoint that commenced when the log was last switched has not completed.
- Log file switch (archiving needed). A redo log cannot be used because an archive operation that commenced when it was last switched has not completed.

These waits may indicate that:

- Your redo logs are on slow devices.
- Your LOG_BUFFER setting is too low.
- You have too few or too small redo logs.

If you are getting log file switch (archiving needed) events, consider alternating your redo logs across multiple devices to reduce contention between the log writer and archiver processes.

For more information see Improving redo log writer performance on page 13.

Buffer busy waits

Buffer busy waits occur when a session cannot access a needed block because it is in use by another session. The two most common causes are insufficient free lists for a table or too few rollback segments.

If buffer busy waits are significant, check the category of events being waited for (on the **Wait activity** tab.

If most buffer waits are for data blocks, then it is likely that you need to create multiple FREELISTS (using the FREELIST clause of the CREATE TABLE statement) for tables that are subject to very heavy concurrent inserts. If the leading category is for Undo Header or Undo Block, you may need to create additional rollback segments.

Free buffer waits

Free buffer waits occur when a session needs to read a data block from a database file on disk into the buffer cache. If there are no unmodified (or clean) blocks in the buffer cache then the session has to wait for the Database Writer process to write modified (or dirty) blocks to disk in order for free buffers to be made available.

Normally, the Database Writer is constantly writing dirty buffers to disk and so this event rarely occurs. When it does occur, it usually due to one of the following reasons:

- Untuned disk layout. If datafiles are not spread evenly across disk devices then a single disk may form a bottleneck to both read and write performance. In this circumstance, the Database Writer may not be able to clear dirty blocks from this device as rapidly as they are created.
- Untuned Database Writers. To efficiently write to multiple disk devices, it is essential that you either configure multiple Database Writers or implement asynchronous or list I/O. This helps the Database Writer to keep up with changes to the buffer cache.
- Untuned sorts. If the SORT_DIRECT_WRITES parameter is set to FALSE, large sorts that require a temporary segment may write the sort blocks to the buffer cache and rely on the Database Writer to move them into the temporary segment. This can flood the buffer cache and cause other sessions to encounter free buffer waits.

Write complete waits

This wait occurs when a session tries to modify a block that is currently being written to disk by the Database Writer process. This happens occasionally, but if it is contributing significantly to overall waits it may indicate inefficiencies in the Database Writer.

The solution may involve optimizing datafile I/O and Database Writer configuration. This can be done by spreading datafiles across multiple disks, using multiple Database Writers, or employing asynchronous or list I/O.

Enqueue waits

Enqueue waits occur when a session waits to obtain a lock. In most cases, this occurs because of a lock on a table or row that the waiting session needs to lock or modify. In some circumstances, the lock involved may be an Oracle internal lock. If the database is well tuned and the application design sound, enqueue waits should be negligible. Common causes of excessive enqueue waits are:

- Contention for a specific row in the database. The application design may require that many processes update or lock the same row in the database. Once common example of this occurs when primary keys are generated using a sequence table.

- Table locks caused by foreign keys that have not been indexed. If an update is made to a foreign key that has not been indexed, the parent table is subjected to a table lock until the transaction is complete.
- Old-style temporary tablespaces. If the tablespace nominated as the temporary tablespace has not been identified with the TEMPORARY clause (introduced in Oracle 7.3), sessions may contend for a space transaction lock.
- The space reserved for transactions within a data block is too small. By default, only one transaction slot for tables or two for indexes is allocated when the table or index is created. If additional transaction slots are required they are created, providing there is free space in the block. However, if all transaction slots are in use and there is no free space in the block, a session that needs to lock a row in the block, encounters an enqueue wait. This occurs even if the row in question is not actually updated or locked. This can occur if both PCTFREE and INITRANS were set too low.

Latch free waits

Latches are Oracle internal locking mechanisms. They prevent multiple sessions from simultaneously updating the same item within Oracle shared memory (SGA). If a session needs to acquire a latch that is held by another session, a latch free wait may occur.

The presence of latch free waits of any significant magnitude may indicate a bottleneck within the SGA. The specific action depends on the latch. For more information see Relieving latch contention on page 28.

4

Using Explain Plan

The **Explain Plan** command allows you to determine the execution plan Oracle applies to a particular SQL statement. Instance Monitor allows you to view graphical representations of the execution plans for SQL statements being executed by a user or those identified by the Top SQL drilldown.

For more information see the Explain Plan window on page 43.

Interpreting the execution plan

Interpreting a formatted execution plan requires practice and often some degree of judgment. However, the following fundamental principles guide the interpretation:

1. The more heavily indented an access path is, the earlier it is executed.
2. If two steps are indented at the same level, the uppermost statement is executed first.
3. An access path may be comprised of a number of steps in the execution plan. For instance, an index access is shown as an INDEX SCAN together with a TABLE SCAN BY ROWID. In this case, the indentation level of the outermost access determines the precedence of the execution. For instance, in the Explain Plan above the most heavily indented operation is the index range scan of an EMPLOYEES index. However, this operation is combined with the ROWID access of the EMPLOYEES table. Therefore the CUSTOMERS access is the first step executed.

Common execution steps

The following tables describe some of the common execution steps that you may encounter:

- Table Access paths (page 44)
- Index Operations (page 44)
- Join Operations (page 44)
- Set Operations (page 45)
- Miscellaneous (page 46)
- Aggregation (page 46).

The steps are defined by the combination of the OPERATION and OPTION columns of the PLAN_TABLE.

Explain Plan window

To display the **Explain Plan** window, right click in the **Top SQL** drilldown. The Explain Plan window displays with its own toolbar. The toolbar contains the following icons:

Describe the database object in the step currently highlighted.

Display a diagrammatic representation of the explain plan.

Display an English language representation of the explain plan.

Only one of the above three icons can be selected.

Show or hide the SQL window.

Rerun the Explain plan.

Table Access paths

The following table shows some of the common table access paths you may encounter while using the Explain plan:

Operation	Option	Description
TABLE ACCESS	FULL	The well known full table scan. This involves reading every row in the table (that is, every block up to the high water mark).
TABLE ACCESS	CLUSTER	Access of data via a index cluster key.
TABLE ACCESS	HASH	Using a hash key to access one or more rows in a table with a matching hash value.
TABLE ACCESS	BY ROWID	Access a single row in a table by specifying its ROWID. ROWID access is the fastest way to access a single row. Often the ROWID is obtained by an associated index lookup.

Index Operations

The following table shows some of the common index operations you may encounter while using the Explain plan:

Operation	Option	Description
AND-EQUAL		The results from one or more index scans are combined.
INDEX	UNIQUE SCAN	An index lookup that returns the address (ROWID) of only one row
INDEX	RANGE SCAN	An index lookup that returns the ROWID of more than one row. This can be because the index is non-unique or because a range operator (for example, >) was used.

Join Operations

The following table shows some of the common join operations you may encounter while using the Explain plan:

Operation	Option	Description
CONNECT BY		A hierarchical self-join is performed on the output of the preceding steps.
MERGE JOIN		A merge join performed on the output of the preceding steps.
MERGE JOIN	OUTER	An outer join implemented by a merge join
NESTED LOOPS		A Nested Loops join is performed on the preceding steps. For each row in the upper result set, the lower result set is scanned to find a matching row.
MERGE JOIN	SEMI	Denotes a sort-merge semi-join.
HASH JOIN	SEMI	Denotes a hash semi-join.
MERGE JOIN	ANTI	Denotes a hash anti-join.
NESTED LOOPS	OUTER	An outer join implemented by a nested loop join.

Set Operations

The following table shows some of the common set operations you may encounter while using the Explain plan:

Operation	Option	Description
CONCATENATION		Multiple result sets are merged in the same way as in an explicit UNION statement. This typically occurs when an OR statement is used with indexed columns.
INTERSECTION		Two result sets are compared and only rows common to both are returned. This operation usually only takes place as a result of an explicit use of the INTERSECTION clause.
MINUS		All result sets in the first result set are returned, except those appearing in the second result set. This occurs as a result of the MINUS set operator.
UNION-ALL		Two result sets are combined and rows from both returned.

Operation	Option	Description
UNION		Two result sets are combined and rows from both are returned. Duplicate rows are not returned.
VIEW		Either a view definition has accessed or a temporary table has been created to store a result set.

Miscellaneous operations

The following table shows some of the miscellaneous operations you may encounter while using the Explain plan:

Operation	Option	Description
FOR UPDATE		The rows returned are locked as a result of the FOR UPDATE clause.
FILTER		Rows from a result set that do not match a selection criteria are eliminated.
REMOTE		An external database is accessed through a database link.
SEQUENCE		An ORACLE sequence generator is accessed to obtain a unique sequence number.
SORT	ORDER BY	A result set is sorted in order to satisfy an ORDER BY clause.

Aggregation operations

The following table shows some of the aggregation operations you may encounter while using the Explain plan:

Operation	Option	Description
COUNT		Count the rows in the result set in order to satisfy the COUNT() function.
COUNT	STOPKEY	Count the number of rows returned by a result set and stop processing when a certain number of rows are reached. This is usually result of a WHERE clause which specifies a maximum ROWNUM (for instance WHERE ROWNUM <= 10).

Operation	Option	Description
SORT	AGGREGATE	This occurs when a group function is used on data which is already grouped. For instance SELECT MAX(AVG(SALARY)) FROM EMPLOYEES GROUP BY MANAGER_ID.
SORT	JOIN	Sort the rows in preparation for a merge join.
SORT	UNIQUE	A sort to eliminate duplicate rows. Typically occurs as a result of using the distinct clause.
PARTITION	SINGLE	Denotes that operations were performed on a single partition.
PARTITION	CONCATENATED	Denotes that operations were performed on multiple partitions.
LOAD	AS SELECT	Denotes an insert append operation.
SORT	GROUP BY	A sort of a result set in order to group them for the GROUP BY CLAUSE.

A

Bibliography

This bibliography lists references that might help you tune and configure your Oracle database:

- *Oracle Performance Tuning*, Second Edition. Mark Gurry and Peter Corrigan, 1-56592-237-9, O'Reilly & Associates, 1996.
- *Oracle 8 Advanced Tuning and Administration*, Eyal Aranoff, Kevin Loney, Noorali Sonawalla, Oracle Press, Osborne McGraw-Hill, 1998
- *Oracle SQL High Performance Tuning*, Guy Harrison, 0-13-614231-1, Prentice Hall, 1997
- *Oracle Server Reference Manual*, Oracle Corporation. Available on your Oracle Documentation CD.
- *Oracle Server Tuning*, Oracle Corporation. Available on your Oracle Documentation CD.
- *Oracle Concepts*, Oracle Corporation. Available on your Oracle Documentation CD.

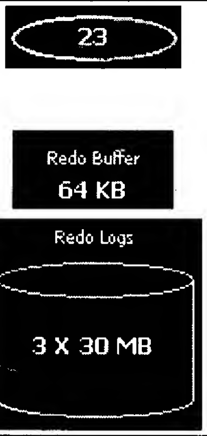
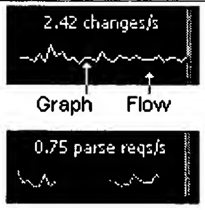
Glossary

Activity button	Click the Activity button to display the activity drilldowns. These include: • Activity Summary tab • Wait activity tab • Lock activity tab • Latch activity tab • Server activity tab.
aggregate operations	Operations that group related rows and return a single row for each group. For example, returning the total number of employees in each department. Aggregate operations are invoked with the group by operator.
Alarm Log button	Click the Alarm Log button to display the Alarm log. The Alarm log contains information about the alarms that have been raised in this Instance Monitor session.
archiver process (ARCH)	This Oracle process copies completed redo logs to backup storage.
array processing	Allows a single SQL call to process multiple rows. For example, a single execution of an insert statement could add multiple rows, or a single fetch from a select statement could return multiple rows. In programming environments, array variables are used to hold the rows fetched or inserted. In many development and inquiry tools, array processing is enabled transparently and automatically.
artificial key	A unique key that contains no real-world information. Artificial keys are usually generated using Oracle sequences. Compare with natural key.

asynchronous	An asynchronous call is one that can return control before the operation has completed. Asynchronous IO allows a process to queue requests to multiple devices concurrently. Asynchronous IO allows a process to submit multiple IO requests without waiting for each request to complete. In practice, this means a single process can utilize the bandwidth of multiple disks.
B*-tree index	An index structure which takes the form of a hierarchy or inverted tree. This is the default format for Oracle indexes.
background processes	Perform specialized tasks on behalf of all sessions. For example, the Database Writer (DBWR) is responsible for writing changed blocks from the buffer cache to the database files. The log writer (LGWR) is responsible for writing blocks from the redo buffer to the redo logs. The archiver process (ARCH) copies completed redo logs to backup storage. Other processes (such as SMON and PMON) perform housekeeping functions, and some processes can only be enabled if certain Oracle options are enabled.
binary chop	A procedure for searching a sorted list of items. The list is successively divided into two sections and the section that must contain the desired item further sub-divided. Eventually the remaining portion is sufficiently small to enable a sequential scan. This technique is useful in programs that cache table data to avoid excessive database access.
bind variables	Allow the variable portions of an SQL statement, such as data values, to be inserted or a search key, to be defined as "parameters" to the SQL statement. The use of bind variables allows SQL statements to be re-executed without re-parsing the SQL statement. The alternative approach, where substitution variables are embedded as literals within the SQL statement requires that the SQL statement be re-parsed when re-executed.
block	The basic unit of storage in an Oracle instance. Block sizes most commonly range between 2 and 32 KB.
branch blocks	The middle level of blocks in a B-tree index. Each branch block contains a range of index key values and pointers to the appropriate leaf blocks.

browse button	Click the **Browse** button to search for a file or to specify the drive and directory where a file is to be saved. When you click this button, the **Open** window appears. Use the standard Windows commands to locate the file or directory. When you have found the file or directory, click **Open**. The file and pathname are shown in the field where you clicked the **Browse** button.
buffer cache	An area in the SGA that contains copies of blocks from database files. The buffer cache exists primarily to reduce disk I/O. It does this by allowing sessions to access frequently or recently accessed data in memory.
cache buffer lru chain	The cache buffer lru chain latch protects the LRU list. The LRU list records how recently a block was accessed.
calibration	Determines the maximum and minimum values for every dataflow by observing the data moving through the database system. This information helps Instance Monitor draw the dataflows correctly. You can manually override these calibrated thresholds at any time, and for any given dataflow.
cardinality	A measure of the number of unique values within a column or an index. The higher the cardinality of the index, the fewer the number of rows that are returned by an exact lookup of a key value (and hence the more useful the index).
chart	A graphical representation of a statistic over a period of time. One or more statistics may be shown on the same graph. To highlight an area of the chart, position your cursor at the top left corner of the area you want to highlight. Click and hold the left mouse button and drag it across the area you want to highlight. An outline is shown around the area you selected. When you release the mouse button, the graph is redrawn showing just the area you highlighted. You can continue to zoom into the area you have highlighted. To display the graph at normal size, click and hold the mouse button in the graph. Move the cursor up and to the left. When you release the mouse button the graph is redrawn at normal size.

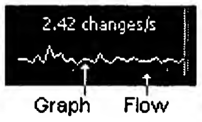
checkpoint	The process of writing all modified blocks in the buffer cache to disk.
client	A software application that requests the services, data, or processing of another application or computer (known as the server).
column	A portion of a database table that stores a particular type of information. When a column is defined, it is given a name and datatype. A table of addresses, for example, might contain a column called CITY and another called TELEPHONE NUMBER.
component	The icons and dataflows shown on the main **Instance Monitor** window. The following diagrams show examples of components.

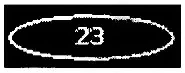
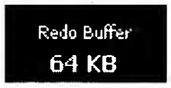
Icons	Dataflows
	

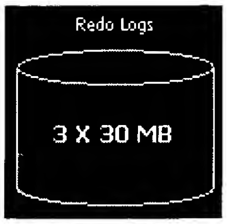
See also dataflow, icon, and label.

concatenated index	An index which is comprised from more than one column.
connect string	The string that is used to link to a database. The database name is defined within Oracle utilities.
connectivity software	A program that is used to establish and maintain a connection between a database and a client application. Once a connection is established, the client can access, modify, and store data on the database. Under Oracle, you can use SQL*Net or Net8 (for Oracle 8) to establish a connection to a database. Under SQL Server, you can use the ODBC drivers provided with Windows.

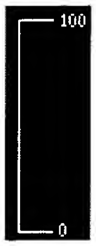
consistent read	Oracle queries return rows that are consistent with the time at which the query commenced. This consistent read may require access to rows that have changed since the query commenced - these rows are accessed from rollback segments.
cost-based optimizer	Determines the execution plan based on an estimate of the computer resources (the cost) required to satisfy various access methods. The cost-based optimizer uses statistics (including the number of rows in a table and the number of distinct values in indexes) to determine the optimum plan.
cursor	A memory structure that contains an SQL statement or PL/SQL block. Also referred to as a context area.
database connection	A link to a database and its data. This link is established when you log on to the database.
database files	Files that contain the data that makes up an Oracle database.
database object	Anything that is stored in a database, including tables, views, indexes, snapshots, triggers, and stored program units.
database segments	Include user objects such as tables and indexes, as well as rollback segments and temporary segments. A segment can belong to only one tablespace.
database writer (DBWR)	The process that is responsible for writing changed blocks from the buffer cache to the database files.
dataflow	A line graph on the main **Instance Monitor** window. Dataflows depict the flow of information between different components of the database management system. The color of a dataflow can change in response to the data that is displayed. You can display a dataflow as a pulse or as a flow and a graph. For more information see flow, graph, and pulse.
DBA	Database Administrator. The person who maintains the databases in your organization.
dedicated server	Processes that perform work on behalf of a single client process. The number of dedicated servers varies as users log in and out of the database.
denormalization	The process of re-introducing redundant or derived information into a data model with the aim of improving performance.

driving table	The table that is accessed first in a table join. Choosing the best driving table is a key decision when optimizing join order.
extent	Segments are composed of a number of distinct storage allocations known as extents. As a segment grows it allocates extents as required, up to the limit of available space and the value of maxextents.
flow	The **flow** shows you the current level of activity. As the rate of data transfer increases, so too does the speed of the flow. If the statistic represented by the flow moves into another threshold, the flow may change color. The combination of movement and color makes it easy to spot congested areas. The **graph** sits on top of the flow and shows you how the load has varied over time. The following diagram shows an example of a flow and graph. 
foreign key	A column or columns within one table which relate to the primary key of a master or parent table. These matching foreign and primary key columns can be used to join the two tables.
free buffer waits	Waits that occur when a session wants to read a data block from a database file on disk into the buffer cache. If there are no unmodified (or clean) blocks in the buffer cache, the session has to wait for the Database Writer process to write modified (or dirty) blocks to disk in order for free buffers to be made available.
free lists	A free list is a list of blocks that are eligible for insert. Each segment contains at least one freelist. Multiple freelists can be configured using the freelists clause if the segment is subject to high concurrent insert rates. Multiple freelist groups can be configured in an Oracle Parallel Server environment so that each instance inserts rows into specific blocks.

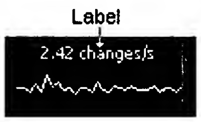
graph	A white line that sits on top of a pulse. The graph represents how the load on the database has varied over time. See flow for more information.
hash value	Hashing refers to the technique of mathematically transforming a key value into a relative address which can be used to rapidly locate record. This relative address is known as a hash value. Oracle uses hashing as a table access method (hash clusters) and to optimize certain join operations (hash join). Hashing is also used extensively within internal SGA operations.
hashing	In general, hashing refers to the technique of mathematically transforming a key value into a relative address that can be used to rapidly locate records. Oracle uses hashing as a table access method (hash clusters) and to optimize certain join operations (hash join). Hashing is also used extensively within internal SGA operations.
hierarchical queries	A special case of a self-join in which each row accessed child rows in a hierarchy of parent-child relationships. This is sometimes referred to as explosion-of-parts.
high water mark	The highest block in a segment that has ever contained data. The high water mark increases as rows are inserted into the segment. Deleting rows does not reduce the high water mark. Full table scans access all rows in the segment up to the high water mark.
I/O	Input or output to a peripheral device. In an Oracle context, I/O refers to input or output disk devices.
icon	The icons in Instance Monitor fall into the following categories:  Process icons are oval in shape and contain a single value that represents the state or existence of a database process.  Memory icons are rectangular and show the utilization of database-specific areas in memory.



Disk icons are cylindrical and fill up as a file increases in size.



Meters show a measurement. The highest and lowest possible values of the measurement are shown.

IO button	Click the IO button to display the I/O drilldowns. These include: • I/O Summary tab • I/O by datafile tab • Logical I/O tab.
label	Labels are shown above most icons and dataflows.  A **label** may have different metrics and thresholds to the component it is over. You can also tailor the metrics and thresholds of the labels.
latch	An internal Oracle locking mechanisms. Latches prevent multiple sessions from simultaneously updating the same item within Oracle shared memory (SGA).
latch free wait	The wait that occurs when a session needs to acquire a latch that is held by another session.
latch miss	If a process requires a latch and cannot obtain it on the first attempt, a latch miss results. The session repeatedly attempts to obtain the latch up to value of the configuration parameter SPIN_COUNT.
leaf blocks	The lowest level of blocks in a B-tree index. Each leaf block contains a range of index key values and pointers (ROWIDs) to appropriate blocks.

least recently used (LRU) list	The least recently used (LRU) algorithm is used by Oracle to remove cached data blocks that have least recently been accessed. When a block is read from disk, it is placed on the Most Recently Used end of the LRU list, unless it has been read in from a table scan of a large table and the CACHE hint has not been specified.
list I/O	List I/O allows multiple IO requests to be submitted in a single call. It is similar to asynchronous IO.
log file parallel write event	The log file parallel write wait event occurs when the log writer waits for IO to redo log devices to complete.
log file sync	The log file sync wait event occurs when a session issues a commit and must wait for a redo log IO before completing.
log writer (LGWR)	The Oracle process that is responsible for writing blocks from the redo buffer to the redo logs.
main menu bar	The menu bar is shown at the top of the **Instance Monitor** window. It contains the following options. They are: • Monitor • Navigator • Tools • Help.
metric	A unit of measurement that can be applied to a database. Metrics can help you gauge the performance of a database system.
multi-threaded servers	See server processes.
natural key	A unique identifier for a table that is composed of naturally occurring columns in the table. Compare with artificial key.
nested loops join	A join method in which each row of the outer table is read. For each row, a lookup of the inner table is undertaken. This best suits joins where the inner table is accessed via an index lookup.
NULL values	Indicates that a value is missing, unknown, or inapplicable. The use of null values extends the normal two-valued logic to a three-valued logic. Null values are important in SQL tuning because they are not generally stored in indexes and therefore present unique tuning problems.

OLAP	On-line Analytical Processing. Involves the real-time manipulation of large quantities of data generally for the purpose of facilitating business decisions. OLAP databases are typified by large data volumes and infrequent, long running queries.
OLTP	On-line Transaction Processing. OLTP databases typically have a very high rate of update and query activity. OLTP is typified by high rates of index lookups, single-row modifications, and frequent commits.
optimistic locking strategy	A locking strategy based on the assumption that a row is unlikely to be changed by another session between the time the row is queried and the time it is modified. Optimistic locking minimizes the lock duration but requires that the transaction be aborted if the row is changed by another session.
optimizer	The component of the Oracle software that determines the execution plan for an SQL statement. Oracle supports two approaches to query optimization. They are: • The rule-based optimizer determines the execution plan based on a set of rules. The rules rank various access paths. For example, an index-based retrieval has a lower rank than a full table scan. A rule-based optimizer uses indexes wherever possible. • The cost-based optimizer determines the execution plan based on an estimate of the computer resources (the cost) required to satisfy various access methods. The cost-based optimizer uses statistics (including the number of rows in a table and the number of distinct values in indexes) to determine the optimum plan.
Oracle System Global Area	See System Global Area (SGA).
outer table	The first table processed in a join of two tables.
page	A connection to a database.
panel	A group of related components (normally icons) on the main **Instance Monitor** window. The name of the panel is normally shown at the top of the panel.
parallel execution	The execution of an SQL operation using multiple processes or threads. This allows some of the stages of execution to be executed simultaneously and large tables to be scanned by multiple processes.

parsing	The process of preparing a SQL statement for execution. This involves checking the statement for syntax errors, checking for a matching statement in the shared pool, and determining the optimal execution plan. Parsing can contribute significantly the processing overhead, especially in OLTP-like environments.
pessimistic locking strategy	A locking strategy based on the assumption that a row might be changed between the time it is fetched and the time it is updated. Pessimistic locking involves locking the row when it is selected to prevent any concurrent updates.
pinned	The process of accessing a block in the buffer cache.
positioning buttons	Instance Monitor provides two buttons you can use to change the order of thresholds and severities. The buttons are: Click this button to move the item up in the list. For example, to move it from position 3 to position 1. The other items in the list are renumbered. Click this button to move the item down in the list. For example, to move it from position 2 to position 6. The other items in the list are renumbered.
primary key	A column or columns that uniquely identify a row in a table.
process	A unit of execution in a multi-processing environment. A process typically executes a specific program and has a unique and private allocation of memory. The operating system determines the process's access to resources such as CPU, physical memory, and disk.

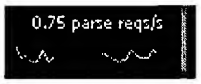
pulse

The **pulse** moves in the direction of the dataflow. As the rate of data transfer increases, so too does the speed of the pulse.

The pulse can change color if the statistic represented by the pulse moves into another threshold.

The combination of movement and color makes it possible to identify congested areas quickly.

The following diagram shows an example of a pulse with a label:



query

A SQL statement that returns a set of values from one or more tables in the database. Instance Monitor uses a variety of queries to collect information about a database's performance.

pause button

Click the Pause button to stop Instance Monitor collecting information.

To restart the data collection, click the button again.

positioning buttons

Instance Monitor provides two buttons you can use to change the order of thresholds and severities. The buttons are:

<<	Click this button to move the item up in the list. For example, to move it from position 3 to position 1. The other items in the list are renumbered.
>>	Click this button to move the item down in the list. For example, to move it from position 2 to position 6. The other items in the list are renumbered.

RAID	Redundant Array of Inexpensive Disks. RAID is used to describe the configuration of multiple physicals disks into one or more logical disks. There are three main types of RAID. They are: • **RAID 0**. Sometimes referred to as striping disks. In this configuration, a logical disk is constructed from multiple physical disks. • **RAID 1**. Referred to as disk mirroring. In this configuration, a logical disk is comprised of two physical disks. In the event that one physical disk fails, processing can continue using the other physical disk. • **RAID 5**. Stripes data across multiple drives while storing sufficient parity information on all drives to allow data to be recovered should any single drive fail.
random I/O	I/O In which a specific disk block is directly accessed. This is typical of the I/O that results from indexed lookups.
redo allocation	The redo allocation latch must be acquired before a session can allocate space in the redo log buffer.
redo buffer	Contains redo log entries that have not yet been written to the redo logs. The redo buffer is periodically flushed and is always flushed when a commit occurs.
redo copy latch	One of the redo allocation latches must be acquired before a session can copy an entry into the redo log buffer.
redo logs	Oracle files that are used to record all changes made to objects within a database. When a commit is issued, the changes made within the transaction are recorded in the redo log. The redo log can be used to restore the transaction in the event of a system failure.
referential integrity	Referential integrity ensures that foreign keys correctly map to primary keys. A referential constraint prevents the insert or update of foreign keys for which there are no matching primary keys. It either prevents the deletion of primary keys if foreign keys exist or deletes these foreign key rows (delete cascade). Referential integrity can result in table-level locking if there are no indexes on the foreign keys.

result sets	The output from a SQL query. Results sets have the same tabular construction as tables. Results sets are also created during intermediate SQL operations. For instance in a multi-table join, each successive join creates a result set. The final result set is the output of the query.
rollback segments	Segments that store the contents of a row before it is modified by a DML (update, insert delete) statement. This information is used in the event of a rollback, to provide a consistent view of the table for queries that commenced before the transaction was committed and to record the eventual success of a transaction.
row level locking	In general, Oracle only ever locks a row that is modified by a DML statement. Page, block, or table locks are not normally applied and read locks are never applied.
ROWID	The ID that uniquely identifies a row by its physical location. The ROWID of a row (if known) is the fastest way to access a row. An index contains the ROWIDs that match specific key values, thus providing quick access to these rows.
rule-based optimizer	The rule-based optimizer determines the execution plan based on a set of rules. The rules rank various access paths. For example, an index-based retrieval has a lower rank than a full table scan. A rule-based optimizer uses indexes wherever possible.
segment	An object in an Oracle database that consumes storage. Examples are tables, indexes, rollback segments, temporary segments, and clusters.
selectivity	A measure of the number of table entries for each index key. The less rows in the table that match specific index keys, the more selective is the index.
sequence table	A sequence table is a table containing sequence number information and is an alternative to using Oracle sequence generators.
sequential I/O	I/O in which disk blocks are read in sequence. This is typical of the I/O that results from full table scans.
serial execution	The execution of an SQL statement using a single process or thread. This requires that each stage of the SQL operation be processed one after the other. Compare with parallel execution.

server processes	The process that performs SQL processing on behalf of an Oracle session. A server process can perform processing for a single session only (known as a dedicated server) or for multiple sessions (known as a multi-threaded server). See also shadow process.
severity	Describes the level of importance of a threshold. A severity is user-defined and determines how Instance Monitor behaves when the values for a metric fall within a range of values. For example, unusually large values might force a metric into a threshold with a high severity. This in turn could change the color of a component, play a sound, or execute an operating-system command.
SGA	See System Global Area.
shadow process	In many environments, the Oracle program runs in a separate process from the client program (for instance, SQL*PLUS). This server process is referred to as the shadow process.
shared pool	An area of the SGA that stores parsed SQL statements, data dictionary information, and some session information. The shared pool reduces parse overhead by caching frequently executed SQL statements.
SMP	Symmetric Multi-processing. A SMP machine contains multiple, equivalent CPUs. The SMP architecture dominates mid-range UNIX computers and is increasingly popular on Microsoft NT systems.
spike	An abnormally high maximum value in a dataflow or graph.
spin lock	If a process requires a latch and cannot obtain it on the first attempt, a latch miss results. The session repeatedly attempts to obtain the latch up to value of the configuration parameter SPIN_COUNT. This technique is known as acquiring a spin lock.
SQL button	Click the SQL button to display the SQL drilldowns.
standard deviation	A measure of how widely values diverge from the mean.
striping	A familiar term for RAID 0. Striping involves spreading data evenly across a number of disks, thus allowing higher data transfer rates than would otherwise be possible.
System Global Area (SGA)	An area of shared memory that stores information that can be shared by multiple sessions.

tablespace	A logical structure that contains and groups together the segments (mainly tables and indexes) that make up a database. A tablespace can consist of more than one database file, but any given database file can belong to only one tablespace.
temporary segments	The storage space for data that is needed for large sorts or for the large, intermediate temporary tables created during SQL statement execution.
thread	A unit of execution that shares its memory space with other threads. Threads can be implemented within processes on some systems or may be used in place of processes in others (for instance, in Windows NT).
threshold	A range of values that might be returned by a metric. If the metric falls within this range, Instance Monitor checks the threshold's severity to determine how to behave. For example, the component representing the metric might change color.
Tooltip	A message that appears whenever the mouse cursor moves over certain areas of the screen.
Top Sessions button	Click the Top Sessions button to display the Top Sessions drilldowns. These include: • Session Details tab • Session Waits tab • Session SQL tab • Session Locks tab • Track Session tab • Session Statistics tab.
transaction	A set of DML (update, delete or insert) operations that succeed or fail as a unit. A transaction is successfully terminated by the COMMIT statement or aborted with the ROLLBACK statement.
transaction slot	Each database block contains a number of transaction slots, controlled by the parameters INITRANS and MAXTRANS, sometimes also referred to as Interested Transaction List (ITL) entries. A session must acquire one of these entries to acquire or wait for a row level lock.

Index